SYSTEM AND METHOD FOR INDEXING QUERIES, RULES AND SUBSCRIPTIONS

BACKGROUND OF THE INVENTION

Field of the Invention

[0001] The following discloses a method for indexing continual queries, rules, profiles and subscriptions, where the continual query, rule, profile or subscription can contain at least one interval predicate. Specifically, an interval predicate indexing method is disclosed for fast identification of queries, rules, profiles, and subscriptions that match a given event, condition, or publication.

Description of the Related Art

[0002] The present invention and the various features and advantageous details thereof are explained more fully with reference to the nonlimiting embodiments that are illustrated in the accompanying drawings and detailed in the following description. It should be noted that the features illustrated in the drawings are not necessarily drawn to scale. Descriptions of well-known components and processing techniques are omitted so as to not unnecessarily obscure the present invention. The examples used herein are intended merely to facilitate an understanding of ways in which the invention may be practiced and to further enable those of skill in the art to practice the invention. Accordingly, the examples should not be construed as limiting the scope of the invention.

[0003] Content-based publication/subscription (pub/sub) systems, continual queries, profile-based applications, rule-based monitoring systems, and other information dissemination services in a large-scale distributed environment have become feasible and popular with the advent of the World Wide Web (WWW). Users of such systems and applications can easily set up or subscribe to services with a provider via the Web. These subscriptions, continual queries, profiles, and rules usually are expressed as predicates on a set of attributes. Each predicate involves an attribute, an operator and a value. A predicate represents the conditions,

specifications or constraints expressed by the users. Predicates are used to filter out a large number of incoming events, conditions, or publications so that a user is notified only of those that meet his/her interests or specifications.

[0004] One of the most critical components of supporting large-scale continual queries, content-based pub/sub, or profile-based applications is the fast matching of events against the predicates. A large number of events can occur in a short period of time. Each event must be matched against a large number of predicates, perhaps in the hundreds of thousands or even millions. Hence, an efficient event matching system is needed. Usually, a main-memory based predicate index is required. This index must support dynamic insertions and deletions of predicates, as client interests/constraints are intermittently added into or removed from the system. The search complexity and the storage cost must be minimized. Furthermore, predicates may contain non-equality clauses, such as intervals. Unlike equality predicates, interval predicates are particularly difficult to index in the face of dynamic insertions and deletions.

[0005] An interval predicate index is used to efficiently answer the following question: "What are the predicate intervals in a set $Q = \{I_1, I_2, ..., I_n)$ that cover a data point?" Here, $I_1, I_2, ..., I_n$ are predicate intervals, such as [4, 5], [2, 19], [24, 230] or [-, 8], that are specified by queries, rules, profiles or subscriptions. These predicate intervals represent the ranges of data values that users are interested in. The problem is to efficiently find all the queries or rules that a given data satisfy or match by maintaining an efficient interval index on the queries, rules or subscriptions. There are some systems in the area of interval indexing. However, they are mostly not effective for fast matching of events in a large-scale dynamic environment. Segment trees and interval trees (H. Samet, Design and Analysis of Spatial Data Structure, Addison-Wesley, 1990) generally work well in a static environment, but are not adequate when it is necessary to dynamically add or delete intervals. Originally designed to handle spatial objects, such as rectangles, R-trees (A. Guttman, "R-trees: A dynamic index structure for spatial searching," *Proceedings of the ACM SIGMOD*, 1984) can be used to index intervals. However, when there is heavy overlapping among the intervals, the search time can quickly degenerate. IBS-trees (E. Hanson, et al., "A predicate matching algorithm for database rule systems," *Proceedings of ACM SIGMOD*, 1990) and IS-lists (E. Hanson, et al., "Selection predicate

indexing for active databases using interval skip lists," *Information Systems*, 21(3):269-298, 1996) were designed for interval indexing. As with most other dynamic search trees, the search time is $O(\log(n))$ and storage cost is $O(n\log(n))$, where n is the total number of predicate intervals. Moreover, in order to achieve the $O(\log(n))$ search time, a complex "adjustment" of the index structure is needed after an insertion or deletion. The adjustment is needed to re-balance the index structure. The adjustment of index increases the insertion/deletion time complexity. More importantly, the adjustment makes it difficult to reliably implement the algorithms in practice. Hence, a need is recognized for a new and more effective interval indexing method.

SUMMARY OF THE INVENTION

[0006] This invention introduces a new concept called virtual construct intervals (VCI), where each predicate interval is decomposed into one or more of these construct intervals. These VCIs strictly cover the predicate interval. Namely, every attribute value covered by the predicate interval is also covered by at least one of the decomposed VCIs, and vice versa. Each construct interval has a unique ID or interval coordinate and a set of endpoints. A construct interval is considered activated when a predicate interval using it in its decomposition is added to the system. The predicate ID is then inserted into the ID lists associated with the decomposed VCIs. To facilitate fast search, a bitmap vector is used to indicate the activation of VCIs that cover an event value. The challenge is to find an appropriate set of construct intervals to make predicate decomposition simple and, more importantly, to build efficient bitmap indexes. Because each construct interval covers only a small range of attribute values, the invention also uses bitmap clipping to cut unnecessary bitmap storage. To facilitate bitmap clipping, the invention introduces the covering segment concept. Bit positions outside a covering segment are pruned.

[0007] The invention supports efficient continual query/rule monitoring, making possible fast matching of a large number of queries, rules or subscriptions. The invention is efficient in both search time complexity and storage requirement for maintaining the query/rule index. The insertion and deletion are also very efficient. The search time complexity for finding all the predicate intervals that cover a data point is $O(1)$, independent of the number of predicate

intervals maintained so far. The storage requirement is $O(n)$, which is proportional to the number of predicate intervals maintained so far. The insertion time is also $O(1)$.

A BRIEF DESCRIPTION OF THE DRAWINGS

[0008] Fig. 1 is a block diagram of a distributed system environment where the current invention may be deployed.

[0009] Fig. 2 is a block diagram of a system that implements the current invention.

[0010] Fig. 3 is an example showing the problem of performing event matching in the presence of one or more interval predicates.

[0011] Fig. 4 is an example of a set of simple construct intervals (SCI).

[0012] Fig. 5 is a flow chart diagram implementing the predicate insertion handler.

[0013] Fig. 6 is a flow chart diagram implementing the event matching handler.

[0014] Fig. 7 is a flow chart diagram for the predicate deletion handler.

[0015] Fig. 8 is an example showing the VCI indexing using simple construct intervals.

[0016] Fig. 9 shows an example of logarithmic construct intervals (LCI).

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

[0017] This invention is related in general to efficient query/rule indexing, where each query or rule may involve one or more interval predicates. Specifically, an efficient interval predicate indexing scheme is disclosed. Such an interval predicate index is used to efficiently answer the following question: "What are the predicate intervals in a set Q = $\{I_1, I_2, ...I_n\}$ that cover a data point?" Here, $I_1, I_2,\ldots,$ In are predicate intervals such as [4, 5], (2, 19), (24, 230) or (-,*), that are specified by queries or rules. These predicate intervals represent the ranges of data values that users are interested in. The problem is to efficiently find all the queries or rules that a given data satisfy or match by maintaining an efficient index on the queries or rules. The invention supports efficient continual query/rule monitoring, making possible real-time monitoring of a large number of queries or rules. The invention is efficient in both search time

complexity and storage requirement for maintaining the query/rule index. The search time complexity for finding all the predicate intervals that cover a data point is O(1), independent of the number of predicate intervals maintained so far. The storage requirement is O(n), which is proportional to the number of predicate intervals maintained so far.

[0018] Fig. 1 is a block diagram of a distributed system environment where various kinds of service providers 104, 105, data sources 102, 103, and clients 100, 101 are interconnected via a communication network 106. The communication network 106 can be the Internet or a wireless network. Various kinds of services, such as pub/sub, continual query monitoring, event monitoring, or rule monitoring, can be offered to clients on the network. The service providers employ the invention disclosed in the current preferred embodiment to monitor and manage data or events coming from one or more data sources 102, 103, in the distributed environment. The data sources can be temperature sensors, traffic sensors, data streams for stock prices, data streams for retail-store sales, and other. The clients 100, 101 can be any device that can receive signals, such as a personal computer, a cell phone, a traditional phone, or a personal digital assistant.

[0019] Fig. 2 is a block diagram of a system that implements the invention disclosed in the current preferred embodiment. It is a computer system 230, which contains at least a CPU 233, a disk 232, and a main memory 230. Contained in the disk 232 and the main memory 231 are various software programs, one of which implements the logic of the invention. The program implementing the disclosed interval indexing method 210 contains at least a predicate insertion handler 211 (further described in Fig. 5), an event matching handler 212 (further described in Fig. 6), a predicate deletion handler 213 (further described in Fig. 7), and a user account manager 214. The user account manager 214 maintains the user subscriptions and other account management tasks. The main logic of the interval indexing method includes the predicate insertion handler 211, the predicate deletion handler 212 and the event matching handler 213, and they will be further described in more detail. The inputs to the system are user interests 201 and event values 202. The user interests are expressed in the forms of interval predicates in the subscriptions, continual queries, and rules. The event values are the readings from sensors or the contents of a publisher. The output 203 of the system is the matched subscriptions, queries or rules.

[0020] Fig. 3 is an example showing the problem of performing event matching in the presence of one or more interval predicates. Predicate intervals 311-319 are drawing horizontally from the left endpoint to the right endpoint. The event matching problem can be solved by drawing a vertical line at the event value. For example, two vertical lines 301, 302 are drawn at a_i and a_j. The matched intervals are those that intersect with the vertical lines. For example, $\{q1, q3, q7, q9\}$ is the set of predicate intervals that match with a_i; $\{q2, q4, q6, q8, q9\}$ is the set of predicate intervals that match with a_j.

[0021] It is quite challenging to quickly identify those predicate intervals intersecting with the vertical line of an event value. Without the help of an interval index, one must perform the event matching via linear search. Namely, each event is compared with all predicate intervals. This can be an unacceptably slow process, especially when the number of predicate intervals is large.

[0022] The disclosed interval indexing method is an attempt to implement the idea shown in Fig. 3 in a cost-effective manner. For simplicity, it is assumed that the endpoints of a predicate interval are integers. However, the attribute can be of integer or non-integer data type. Those skilled in the art will appreciate that the invention can stretch the non-integer endpoints to the nearest integers. A set of virtual construct intervals, VCIs, are pre-defined at the integer values of an attribute. Each VCI has a unique ID, or coordinate, and specific endpoints. These VCIs are used to decompose a predicate interval. The decomposed VCIs strictly cover the predicate interval. Namely, any attribute value covered by the predicate interval is also covered by at least one of the decomposed VCIs, and vice versa.

[0023] For each integer value, the invention uses a bitmap vector to indicate the activation of certain VCIs that cover the value. Each bitmap vector contains a total of $N = RL$ bits, where R is the total number of integer values for the attribute. Associated with each VCI, a predicate ID lists is maintained. The invention uses an array of header pointers to maintain such ID lists.

[0024] Fig. 4 is an example of a set of simple construct intervals (SCI). Under SCI, L VCIs are pre-defined for each integer value. These VCIs have consecutive lengths ranging from 1 to L. The ID of a VCI $c:[a,b]$, where c is the ID and $[a, b]$ is the interval, can be computed as

$c = (a - a_0)L + (b - a) - 1$. In Fig. 4, $L = 4$ and the ID starts from 0 for the interval $[a_0, a_0 + 1]$. There are four VCIs whose left endpoints are at the same integer value. For example, VCIs 0-3 (401-403) all start at a_0, but their right endpoints are different.

[0025] Fig. 5 is a flow chart diagram implementing the predicate insertion handler (211 in Fig. 2). To insert a predicate interval $p:[x, y]$ (step 500), the invention first checks to see if the predicate interval is larger than the largest virtual construct interval $y - x > L$, 501. If not, then no decomposition is needed, and the invention simply finds the ID of the VCI that is the same as the predicate interval (step 505). If decomposition is needed, the invention repeatedly decomposes the predicate interval into $m = \left\lfloor \frac{y - x - 1}{L} \right\rfloor$ intervals of length L, starting from x (step 502). The last remnant is $[x + mL, y]$. After this decomposition, the ID of each decomposed VCI is computed (step 503 and step 504). Finally, for each decomposed VCI c_j, the invention sets the corresponding bitmap if H_{c_j} is empty and insert the predicate ID to the corresponding ID list (step 506). Note that when setting bitmap for c_j, the invention needs to set bitmap b_{i,c_j} to 1 for $a \le i \le b$, where c_j is the ID of $[a, b]$. Here, b_{i,c_j} means the bit position c_j of bitmap vector B_i.

[0026] Fig. 6 is a flow chart diagram implementing the event matching handler (212 in Fig. 2). The search begins with an event value s (step 600). The invention initializes the search result M to empty and t to 0 (step 601). The invention then tests to see if $t < N$, where N is the total number of VCIs (step 602). If not, the search ends and the result is returned (step 603). Otherwise, the invention further tests to see if $b_{s,t} == 1$ (step 604). If yes, then the invention has a matched VCI and its associated ID lists would be combined with M (step 605). After that, the invention processes the next t (step 606) and proceeds to step 602.

[0027] Fig. 7 is a flow chart diagram for the predicate deletion handler (213 in Fig. 2). It is almost the same as the insertion handler, except for step 706. Steps 701-705 are exactly the same as steps 501-505 in Fig. 5 because these steps represent the decomposition process. After decomposition, the invention proceeds to remove the predicate IDs from the ID lists and reset bitmap if the ID list becomes empty after the removal (step 706).

[0028] Fig. 8 is an example showing the VCI indexing using simple construct intervals (SCIs). Two predicates $q:[a_0,a_0+2]$ (801) and $p:[a_0,a_0+6]$ (802) are inserted. The invention uses the example SCI shown in Fig. 4 for decomposition. Each bitmap vector has 28 bits and the invention show 7 bitmap vectors 810-816. An array of 28 pointer header (820) is used to manage the ID lists 821-823. Since $L=4$, no decomposition is needed for predicate $q:[a_0,a_0+2]$. Hence, the ID is inserted into H_1 (821) and $b_{0,1},b_{1,1},b_{2,1}$ are set to 1. On the other hand, $p:[a_0,a_0+6]$ is first broken into $[a_0,a_0+4]$ and $[a_0+4,a_0+6]$. The ID of $[a_0,a_0+4]$ is 3 and that of $[a_0+4,a_0+6]$ is 17. Hence, its ID is inserted into the two ID lists 822-823. The corresponding bit positions are set to 1 as well.

[0029] Thus, the invention provides a method (and service/system) for maintaining and using a query index. As mentioned above, the queries within the query index have predicate intervals (e.g., 311-319 and 801-802). The invention begins by defining groups of virtual construct intervals (400-403, 404-407, 408-411, etc.) and inserting each of the predicate intervals 801-802 into at least one of the groups of the virtual construct intervals. More specifically, predicate interval 801 is inserted into virtual construct interval 401 of the first group (400-403) of virtual construct intervals shown in Figure 4. Virtual construct interval 401 corresponds to bit positions $b_{0,1},b_{1,1},b_{2,1}$ of bitmap vectors 810-812 in Figure 8. The invention only defines virtual construct intervals that are between the minimum and maximum possible attribute values of the predicate intervals.

[0030] Each of the groups of virtual construct intervals (e.g., 400-403) is adapted to hold multiple predicate intervals. Therefore, while virtual construct interval 401 maintains predicate interval 801, virtual construct interval 403 (which is in the same group of virtual construct intervals as a virtual construct interval 401) maintains predicate interval 802. Bit positions $b_{0,3}$, $b_{1,3}$, $b_{2,3}$, $b_{3,3}$, $b_{4,3}$ in bitmap vectors 810-814 (in Figure 8) correspond to virtual construct interval 403 in Figure 4. Thus, event values (a_0, a_0+1, a_0+2, etc.) of the predicate intervals (801, 802) are aligned with the same event values of the virtual construct intervals (401, 403) in that each bitmap vector 810-816 is associated exclusively with a single event value. It necessarily follows that predicate intervals are inserted only into the construct intervals that have corresponding event values. Thus, the same event value (e.g., a_0+1) of multiple predicate intervals (801, 802)

are inserted into the same event value (e.g., a_0+1 in bitmap vector 811) within different virtual construct intervals (401, 403) and simply reside in different bit positions (1, 3) within the given bitmap vector (811). As mentioned previously, the locations of the predicate intervals within the groups of virtual construct intervals are maintained using the predicate ID bitmap vector 820.

[0031] Each of the groups of virtual construct intervals covers a unique group of event values. For example, the first group of virtual construct intervals (400-403) covers event values a_0 to a_0+4, while the second group of virtual construct intervals (404-407) covers event values a_0+1 to a_0+5. All virtual construct intervals in a group (400-403) began at the same attribute value (a_0) and end at different attribute values (a_0+1 to a_0+4). Note that the groups of virtual construct intervals have uniform lengths (Figure 4) , while the predicate intervals have non-uniform lengths (Figure 3). In other words, all of the groups of the virtual construct intervals within the query index have the same pattern of different sized of virtual construct intervals; the first is one interval long, the second is two intervals long, the third is three intervals long, etc. within each of the groups.

[0032] If the predicate interval is small enough, the invention inserts the predicate interval into the same sized virtual construct interval, as shown by inserting the predicate interval 801 into virtual construct interval 401. However, if the predicate interval is larger than any of the virtual construct intervals, the invention first inserts an initial portion of the predicate interval into the largest available virtual construct interval. If necessary, a number of maximum-size virtual construct intervals can be utilized for excessively long predicate interval. The excess length of the predicate interval is referred to as the remnant predicate interval. The invention inserts the remnant predicate interval into the same length virtual construct interval. Thus, the last two event values of predicate interval 802 are placed into a similar length virtual construct interval (which would be virtual construct interval 417 if the repetitions in Figure 4 are repeated).

[0033] Different sets (groups) of VCI can be defined. Fig. 9 shows an example of logarithmic construct intervals (LCI). Instead of consecutive lengths, the invention uses $2^0, 2^1, ..., 2^k$ as the interval lengths. If $L = 4$, the invention uses 1, 2, and 4 different VCI lengths for a set of pre-defined VCIs at an integer value. With logarithmic construct intervals, the total number of VCIs is $R(\log(L)+1)$, which is in general much less than that for the SCI.

[0034] Those skilled in the art will appreciate that some values in a bitmap vector will never be used. This is because some VCIs will never intersect with the vertical line of an event value. For example, [3 7] will never cover any event value less than 2 or greater than 8. Hence, the invention can prune certain bit positions from a bitmap vector. In fact, the invention can identify the minimum ID and the maximum ID for a bitmap vector B_j. For SCI, the minimum ID is $(j-L)L+L-1$ and the maximum ID is $jL+L-1$. Any bit position outside these two boundaries can be clipped, thus saving storage cost.

[0035] Those skilled in the art will also appreciate that user interests may be specified by more than one predicate clauses, some of which may be equality clauses while others non-equality. The interval index disclosed in the current invention can be used to index an attribute with both interval predicates and equality predicates. In this case, the invention treats equality as a virtual construct interval with length of zero. Moreover, separate interval indexes can be used for individual attributes for fast matching of event values for said individual attributes. The final result is derived by combining individual results from separate attributes.

[0036] Those skilled in the art will further appreciate that a predicate interval may be open-ended. In this case, the invention can treat the two open-ended intervals as two special VCIs, one is $[b^{\max},\infty)$, representing the right open-ended interval, and the other is $(-\infty,a^{\min}]$, representing the left open-ended interval. Any event value less than $a^{\min}$ will find all the predicate intervals with $-\infty$ as the left endpoint. Similarly, any event value that is greater than $b^{\max}$ will find all the predicate intervals with ∞ as the right endpoint.

[0037] This invention introduces a new concept called virtual construct intervals, where each predicated interval is decomposed into a set of these construct intervals. Each construct interval has a unique ID or interval coordinated and a set of endpoints. A construct interval is considered activated when a predicate interval using it in its decomposition is added to the system. For each attribute value, a bitmap index is used to indicate the activation of any construct interval that covers the attribute value. The challenge is to find an appropriate set of construct intervals to make predicate decomposition simple and, more importantly, to build efficient bitmap indexes. Because each construct interval covers only a small range of attribute values, using the bitmap clipping to cut unnecessary bitmap storage. To facilitate bitmap

clipping the introduction of the covering segment concept is used. Bit positions outside a covering segment are pruned.

[0038] While the invention has been described in terms of preferred embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.